



ST. ANNE'S COLLEGE OF ENGINEERING AND TECHNOLOGY
(Approved by AICTE, New Delhi. Affiliated to Anna University, Chennai)
Accredited by NAAC
ANGUCHETTYPALAYAM, PANRUTI – 607 106.

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING

Subject Name	: NATURAL LANGUAGE PROCESSING
Subject code	: AL3501
Regulation	: 2021
Year / Branch	: III / AIML
Semester	: V
Faculty Name	: Mr. R. MANICKAVASAGAN, AP/AIML

UNIT III SYNTACTIC ANALYSIS

Context-Free Grammars, Grammar rules for English, Tree banks, Normal Forms for grammar –Dependency Grammar – Syntactic Parsing, Ambiguity, Dynamic Programming parsing – Shallow parsing – Probabilistic CFG, Probabilistic CYK, Probabilistic Lexicalized CFGs - Feature structures,

Unification of feature structures.

Syntactic analysis, also known as parsing, is a crucial phase in Natural Language Processing (NLP) that examines the structure of a sentence to understand its grammatical correctness and relationships between words. It involves analyzing the sentence based on formal grammar rules to determine if it's well-formed and to identify the relationships between words, like subject-verb-object. This process often results in a parse tree or similar hierarchical structure that represents the sentence's syntax.

Here's a more detailed breakdown:

1. Purpose:

- **Grammar Checking:**

Syntactic analysis ensures the sentence adheres to the grammatical rules of the language. For example, it checks for correct word order, subject-verb agreement, and the use of appropriate punctuation.

- **Relationship Identification:**

It helps determine the relationships between words, like which word is the subject, verb, object, or modifier in a sentence.

- **Meaning Extraction:**

While primarily focused on structure, syntactic analysis lays the groundwork for deeper understanding of meaning. By understanding the structure, NLP systems can better interpret the meaning of the sentence.

2. How it works:

- **Formal Grammar Rules:**

Syntactic analysis relies on formal grammar rules (like context-free grammar) to define the structure of the language.

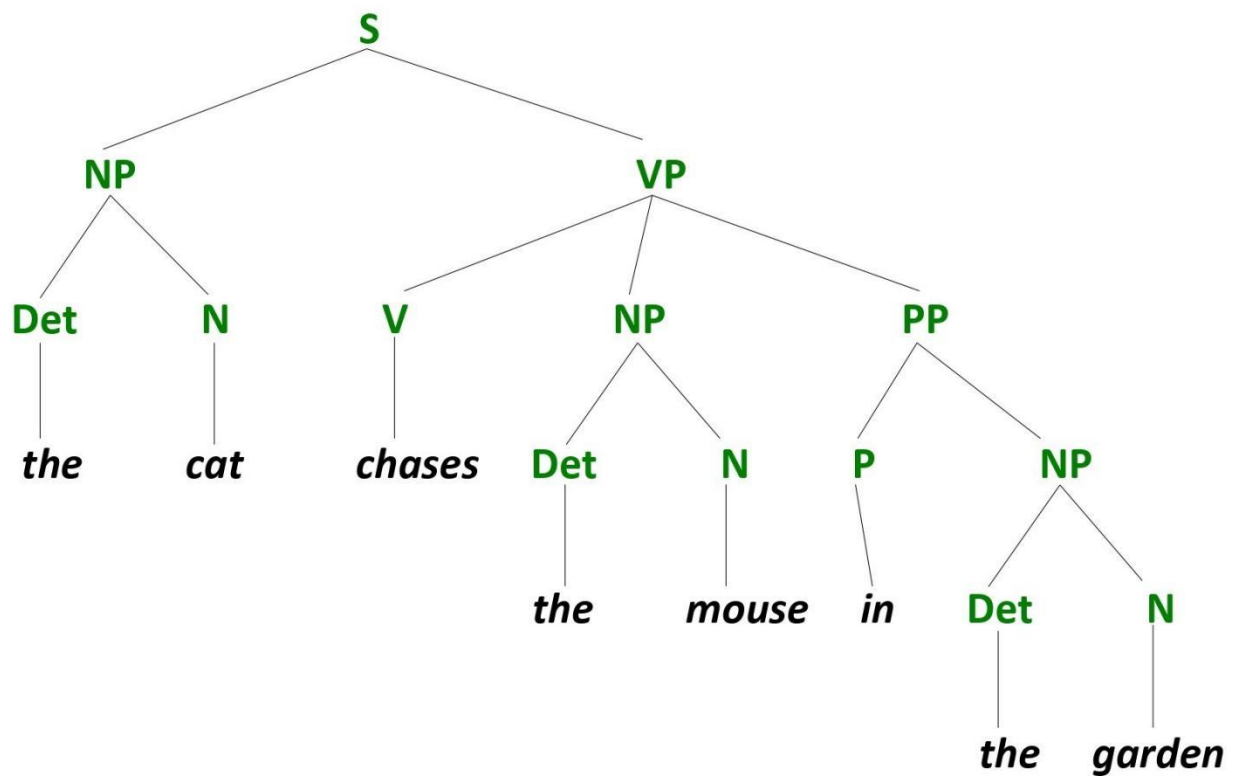
- **Parsing:**

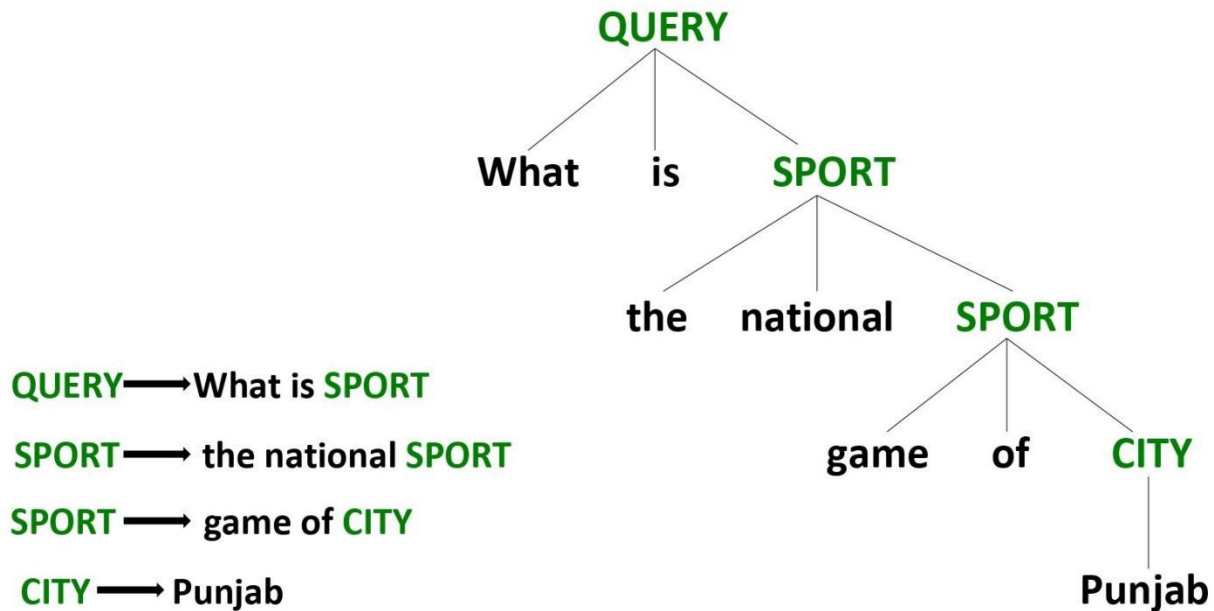
This is the process of analyzing the input string (sentence) against the grammar rules.

- **Parse Tree:**

A parse tree is a common way to represent the hierarchical structure of a sentence, showing how words are grouped into phrases and how these phrases relate to each other.

- **Top-down and Bottom-up Parsing:**





There are different approaches to parsing, including top-down (starting from the root of the tree and building down) and bottom-up (starting from the leaves and building up).

3. Examples:

- **Invalid Sentence:**

A sentence like "Goes the boy school" would be rejected by syntactic analysis because it violates English grammar rules.

- **Valid Sentence:**

A sentence like "The cat chases the mouse" would be analyzed, and a parse tree would be constructed to show the relationships between "cat" (subject), "chases" (verb), and "mouse" (object).

4. Importance in NLP:

- **Foundation for Semantic Analysis:**

Syntactic analysis provides the structure upon which semantic analysis (understanding the meaning of words and sentences) is built.

- **Various Applications:**

It is essential for various NLP tasks, including:

- **Machine Translation:** Understanding the structure of sentences in different languages is crucial for accurate translation.
- **Information Extraction:** Identifying the key entities and relationships in a text (e.g., finding the subject and object in a sentence).

- **Question Answering:** Understanding the structure of questions to find relevant answers.
 - **Text Summarization:** Identifying the most important parts of a text based on its structure.
 - **Chatbots:** Understanding user input to generate appropriate responses.
- In essence, syntactic analysis is a foundational step in NLP, enabling computers to understand the grammatical structure of language and paving the way for deeper semantic understanding and other advanced NLP tasks.

Context-Free Grammars

In NLP, Context-Free Grammars (CFGs) are formal systems used to represent the syntactic structure of sentences. They define rules for how words and phrases can be combined to form valid sentences, aiding in tasks like parsing, language generation, and understanding.

Key Components of a CFG:

- **Non-terminals:**
Represent syntactic categories like noun phrases (NP), verb phrases (VP), etc.
- **Terminals:**
The actual words or tokens of the language.
- **Production Rules:**
Define how non-terminals can be expanded into other non-terminals or terminals (e.g., $S \rightarrow NP VP$).
- **Start Symbol:**
The initial non-terminal from which sentence generation begins (usually 'S' for Sentence).

How CFGs Work in NLP:

1. Parsing:

CFGs are used to analyze the structure of a sentence by breaking it down into its constituent parts (e.g., identifying noun phrases, verb phrases, etc.).

2. Language Generation:

CFGs can be used to generate sentences by starting from the start symbol and recursively applying production rules.

3. Representation:

CFGs provide a formal way to represent the rules of a language, enabling systems to process and understand sentences according to those rules.

Example:

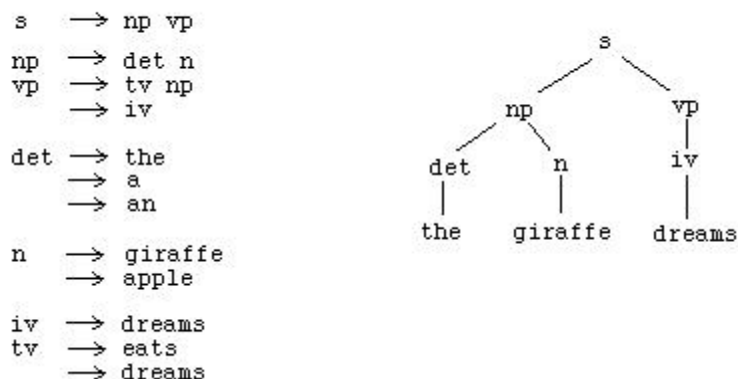
A simple CFG might include rules like:

- S → NP VP (Sentence is composed of a noun phrase and a verb phrase)
 - NP → Det N (Noun phrase is a determiner followed by a noun)
 - VP → V NP (Verb phrase is a verb followed by a noun phrase)
 - Det → "the" | "a" (Determiner can be "the" or "a")
 - N → "cat" | "dog" (Noun can be "cat" or "dog")
 - V → "chases" | "sees" (Verb can be "chases" or "sees")
- This CFG can be used to parse or generate sentences like "The cat chases the dog".

Limitations:

While powerful, CFGs have limitations in handling ambiguity and capturing the nuances of natural language. For more complex NLP tasks, probabilistic models and other techniques are often used alongside or in place of CFGs.

Figure 1. A grammar and a parse tree for "the giraffe dreams".



Grammar rules for English in nlp

In NLP, English grammar rules are used for tasks like parsing and part-of-speech (POS) tagging. These rules define the structure of sentences and how words relate to each other. Key concepts include sentence structure, subject-verb agreement, tense

consistency, and pronoun agreement. Grammars like Context-Free Grammars (CFG) and Dependency Grammars are used to represent these rules.

Here's a more detailed breakdown:

Key Grammar Rules in NLP:

- **Sentence Structure:**

A sentence typically consists of a subject and a predicate (verb and its related elements).

- **Subject-Verb Agreement:**

The verb must agree in number (singular or plural) with the subject.

- **Tense Consistency:**

Verbs should be used consistently within the same tense throughout a sentence or paragraph.

- **Pronoun Agreement:**

Pronouns must agree in number and gender with the noun they replace (the antecedent).

- **Parts of Speech:**

Words are categorized into nouns, verbs, adjectives, adverbs, pronouns, prepositions, conjunctions, and interjections.

- **Modifier Placement:**

Adjectives and adverbs should be placed correctly to modify the appropriate words.

- **Punctuation:**

Proper use of punctuation (like commas, periods, etc.) is crucial for clarity.

- **Word Choice:**

Choosing the right words can significantly impact the meaning and tone of a sentence.

Grammar Formalisms in NLP:

- **Context-Free Grammars (CFG):**

CFGs are a fundamental concept, breaking down sentences into hierarchical structures like noun phrases (NP) and verb phrases (VP).

- **Dependency Grammar:**

This approach focuses on the relationships between words in a sentence, rather than hierarchical structure.

Example:

In the sentence "The cat sat on the mat," the grammar rules would identify:

"The" as a determiner, "cat" as a noun, "sat" as a verb, "on" as a preposition, and "mat" as a noun.

The sentence can be further broken down into a noun phrase ("the cat") and a verb phrase ("sat on the mat"). The verb phrase can be further divided into the verb ("sat") and a prepositional phrase ("on the mat").

NLP systems use these grammatical rules to understand the meaning of sentences, parse them, and perform various tasks like translation, question answering, and information extraction.

Treebanks

In natural language processing (NLP), treebanks are linguistically annotated corpora that represent the syntactic or semantic structure of sentences using tree structures. They are essentially collections of sentences where each sentence is parsed and labeled with its grammatical relationships, forming a hierarchical tree representation. Treebanks are crucial for developing and evaluating NLP systems, particularly in tasks like parsing, machine translation, and speech recognition.

Here's a more detailed explanation:

- **Purpose:**

Treebanks provide a structured way to analyze and understand the grammatical relationships within sentences, enabling NLP models to learn these patterns and improve their performance.

- **Structure:**

Each sentence in a treebank is represented as a tree, where nodes represent words or phrases, and the branches indicate syntactic relationships like subject-verb or noun-adjective.

- **Creation:**

Treebanks are created by linguists who manually annotate sentences with their syntactic structures. This process can be time-consuming and require significant expertise, but it results in high-quality data for NLP training.

- **Applications:**

Treebanks are used to train parsers, which are algorithms that automatically analyze sentence structure. They also serve as benchmarks for evaluating the accuracy of parsers and other NLP systems.

- **Examples:**

A well-known example is the Penn Treebank, which is a large collection of parsed Wall Street Journal articles according to Wikipedia. Other examples include the Sinica Treebank, and the Homer Treebank.

- **Benefits:**

- **Improved Parsing:** Treebanks provide a valuable resource for training and evaluating parsers, leading to more accurate syntactic analysis of sentences.
- **Enhanced NLP Models:** The structured data in treebanks helps NLP models learn grammatical rules and relationships, improving their performance in various tasks.
- **Linguistic Research:** Treebanks facilitate linguistic research by providing a large-scale resource for studying syntactic phenomena and language variation.
- **Benchmarking:** They offer a standard way to evaluate the performance of different NLP systems and algorithms.

- **Construction Process:**

Treebank construction typically involves:

- **Corpus Selection:** Choosing a relevant text corpus (e.g., news articles, books).
 - **Part-of-Speech Tagging:** Assigning grammatical categories (e.g., noun, verb, adjective) to each word.
 - **Parsing:** Analyzing the syntactic structure of sentences and representing it in tree structures.
 - **Annotation:** Adding further linguistic information to the trees, such as semantic roles or dependencies.
 - **Quality Control:** Ensuring the accuracy and consistency of the annotations.
- Key Concepts:**
- **Constituency Parsing:** Represents the hierarchical structure of sentences, showing how words group into phrases and clauses.
 - **Dependency Parsing:** Focuses on the relationships between individual words in a sentence.
 - **Semantic Role Labeling:** Identifies the semantic roles of words and phrases (e.g., agent, patient, instrument).
 - **Part-of-Speech (POS) tagging:** Identifying the grammatical category of each word.

Normal Forms for grammar in nlp

In NLP, normal forms for grammars, like Chomsky Normal Form (CNF) and Greibach Normal Form (GNF), are standardized structures used to simplify parsing and other grammar-related tasks. These forms impose specific constraints on the production rules of a context-free grammar (CFG), making it easier to design and optimize algorithms.

Key Normal Forms:

1. Chomsky Normal Form (CNF):

- All production rules are in one of two forms:
- $A \rightarrow BC$ (where A, B, and C are non-terminal symbols)
- $A \rightarrow a$ (where A is a non-terminal and 'a' is a terminal symbol)
- **Purpose:** CNF simplifies parsing algorithms like CYK (Cocke-Kasami-Younger), providing a uniform structure for rules.
- **Example:** $S \rightarrow AB$, $S \rightarrow a$, $A \rightarrow b$, $B \rightarrow c$.

2. Greibach Normal Form (GNF):

- All production rules are in the form:
- $A \rightarrow a\alpha$ (where 'a' is a terminal, and α is a string of non-terminals)
- **Purpose:** GNF is useful for constructing pushdown automata and can be more efficient for certain types of parsing tasks.
- **Example:** $S \rightarrow aA$, $A \rightarrow bB$, $B \rightarrow c$.

Why Normal Forms?

- **Simplified Parsing:**

Normal forms impose constraints on the structure of production rules, making it easier to design and implement parsing algorithms.

- **Algorithm Optimization:**

Algorithms like CYK are designed to work efficiently with grammars in CNF, leading to optimized parsing performance.

- **Grammar Analysis:**

Normal forms can be helpful in analyzing and understanding the properties of context-free languages.

- **Grammar Transformation:**

Any context-free grammar can be transformed into equivalent grammars in CNF and GNF, allowing for flexibility in choosing the most suitable form for a given task.

Other Normal Forms (Less Common):

- **Backus-Naur Form (BNF):**

A more flexible way to define the syntax of a language, using meta-syntax to represent the grammar.

- **Weak Normal Form (WNF):**

A less restrictive form compared to CNF and GNF, offering a balance between expressiveness and structure.

Dependency Grammar in nlp

Dependency Grammar in Natural Language Processing (NLP) is a framework that represents sentence structure by showing relationships between words. It focuses on the relationships between individual words, rather than grouping them into phrases like constituency grammar. These relationships, or dependencies, are typically represented as directed edges between words in a graph, with arrows indicating the direction of dependency.

Here's a more detailed breakdown:

Key Concepts:

- **Words as Nodes:**

In dependency grammar, each word in a sentence is considered a node in a graph.

- **Dependencies as Edges:**

Relationships between words are represented by directed edges (arrows) connecting the nodes. These edges are labeled to indicate the type of relationship (e.g., subject, object, modifier).

- **Head and Dependent:**

One word in a dependency relationship acts as the "head" (or governor), and the other is the "dependent." The dependent word is modified or related to the head word in some way.

- **Single Root:**

A dependency tree (the graphical representation of a sentence's structure) has a single designated root word, typically the main verb, which is not dependent on any other word in the sentence.

- **No Crossing Arcs (Typically):**

In a projective dependency tree, there are no crossing arcs, meaning the path from the root to any word in the sentence doesn't intersect with paths to other words.
How it Works:

1. Identify Relationships:

Dependency parsers analyze sentences to determine the relationships between words. This often involves identifying grammatical roles like subject, object, verb, and various modifiers.

2. Build the Tree:

Based on these relationships, a dependency tree is constructed, visually representing the sentence's structure.

3. Label Dependencies:

Each dependency arc is labeled with the type of relationship it represents (e.g., "nsubj" for nominal subject, "dobj" for direct object).

Example:

In the sentence "The cat sat on the mat," the dependency parser might identify:

- "sat" as the root word.
- "cat" as the subject of "sat" (nsubj).
- "mat" as the object of the preposition "on" (pobj).
- "the" and "the" as determiners modifying "cat" and "mat" respectively.

Advantages of Dependency Grammar:

- **Simplicity:**

Dependency structures can be simpler and more direct than constituency structures, especially for languages with flexible word order.

- **Efficiency:**

Dependency parsing can be more efficient for certain tasks, such as information extraction and machine translation.

- **Universality:**

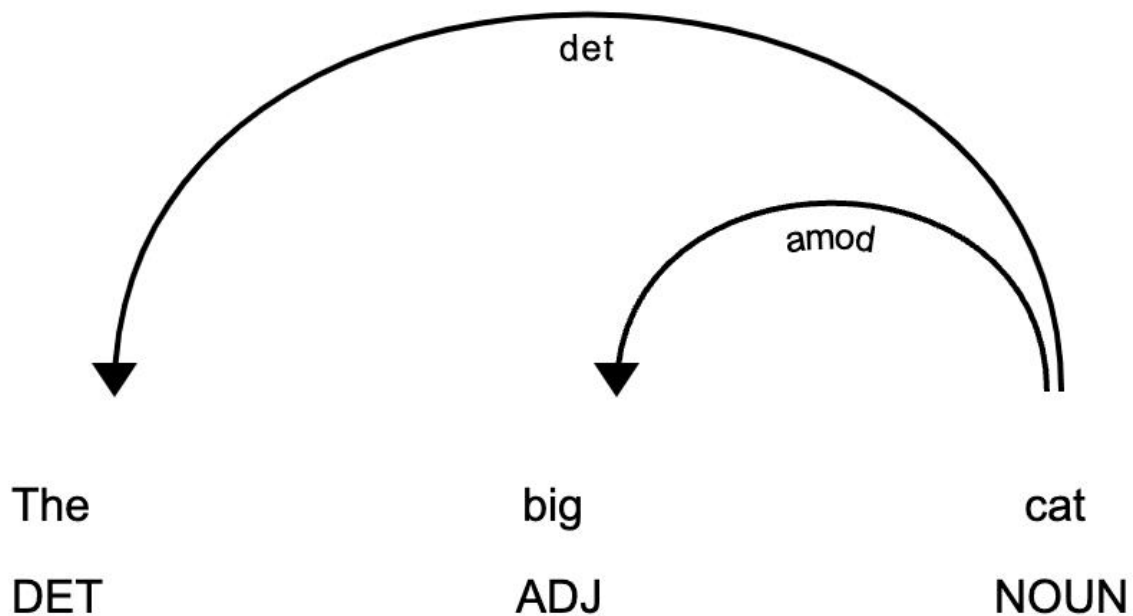
The Universal Dependencies project aims to create a unified dependency framework for various languages, facilitating cross-lingual NLP.

Applications:

Dependency grammar and parsing have various applications in NLP, including:

- **Information Extraction:** Identifying relationships between entities in text.
- **Machine Translation:** Understanding sentence structure to facilitate accurate translation.

- **Question Answering:** Determining the grammatical roles of words to answer questions correctly.
 - **Text Summarization:** Analyzing sentence structure to identify key information.
 - **Coreference Resolution:** Identifying which words refer to the same entities.
- In essence, dependency grammar provides a powerful way to represent and analyze the syntactic and semantic relationships within sentences, making it a fundamental concept in modern NLP.



Syntactic Parsing

Syntactic parsing in NLP is the process of analyzing the grammatical structure of a sentence to understand how its words relate to each other. It essentially breaks down a sentence into its constituent parts (like phrases and clauses) and reveals the relationships between them, often represented as a tree-like structure. This process helps machines understand the grammatical correctness and structural relationships within a sentence, which is crucial for various NLP tasks.

Here's a more detailed explanation:

Key Concepts:

- **Grammar:**

Syntactic parsing relies on a grammar (either context-free grammar or dependency grammar) that defines the rules for how words can be combined into phrases and sentences.

- **Parse Tree:**

The output of syntactic parsing is often a parse tree (or dependency diagram) that visually represents the grammatical structure of a sentence.

- **Constituency Parsing:**

This approach focuses on identifying phrases and their hierarchical relationships (e.g., noun phrase, verb phrase).

- **Dependency Parsing:**

This approach focuses on identifying the grammatical dependencies between individual words (e.g., subject-verb, verb-object).

How it Works:

1. **Tokenization:** The sentence is first broken down into individual words or tokens.
2. **Part-of-Speech Tagging:** Each token is assigned a part-of-speech tag (e.g., noun, verb, adjective).
3. **Parsing:** The parser analyzes the sentence based on the grammar rules and identifies the grammatical relationships between the words.
4. **Parse Tree Generation:** The parser generates a parse tree that represents the hierarchical structure of the sentence.

Why is it important?

- **Understanding Sentence Structure:**

Syntactic parsing helps machines understand the underlying grammatical structure of sentences, which is essential for many NLP tasks.

- **Ambiguity Resolution:**

Natural language is often ambiguous. Syntactic parsing can help resolve some of this ambiguity by revealing the intended grammatical structure.

- **Feature Extraction:**

The parse tree can be used to extract features for other NLP tasks, such as machine translation, text summarization, and question answering.

- **Psycholinguistics:**

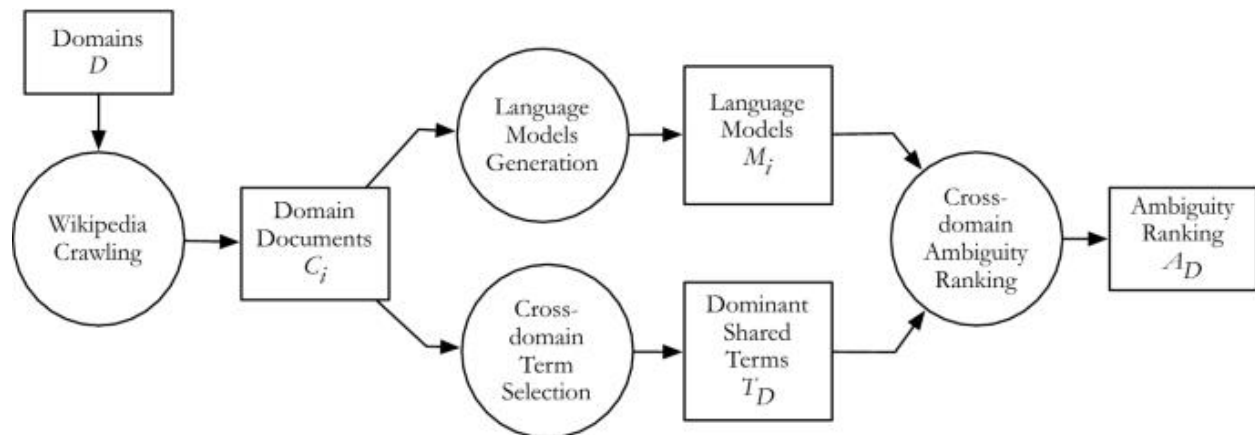
Syntactic parsing is also used in psycholinguistics research to understand how humans process language.

In essence, syntactic parsing is a fundamental component of NLP that enables machines to understand the grammatical structure of sentences, which is crucial for many natural language understanding tasks.

Ambiguity

In natural language processing (NLP), ambiguity refers to the phenomenon where a word, phrase, or sentence can have multiple possible interpretations or meanings. This complexity arises from the inherent flexibility and ambiguity of human language, making it challenging for computers to accurately understand and process text.

Here's a breakdown of key aspects of ambiguity in NLP:



1. Types of Ambiguity:

- **Lexical Ambiguity:**

Occurs when a single word has multiple meanings (e.g., "bank" can refer to a financial institution or the edge of a river).

- **Syntactic Ambiguity:**

Arises from ambiguity in sentence structure, where the same words can be grouped in different ways, leading to different meanings (e.g., "I saw the man with the telescope" - who has the telescope?).

- **Semantic Ambiguity:**

Occurs when the meaning of a sentence is unclear due to the ambiguity of the words or phrases within it (e.g., "The chicken is ready to eat" - is the chicken being served or is it going to eat something?).

- **Pragmatic Ambiguity:**

Relates to ambiguity in the intended meaning based on context and real-world knowledge (e.g., a request like "Can you pass the salt?" is usually interpreted as a polite request, not a question about ability).

- **Anaphoric Ambiguity:**

Arises when pronouns or other referring expressions can refer to multiple entities (e.g., "The horse ran up the hill. It was very steep.").

2. Why Ambiguity is a Challenge in NLP:

- **Computers need precise instructions:**

Unlike humans, computers rely on clear, unambiguous rules and algorithms to process information.

- **Context is crucial:**

Resolving ambiguity often requires understanding the context of a sentence or passage, which can be difficult for NLP models.

- **Intuition and background knowledge are needed:**

Humans can often infer meaning based on intuition and background knowledge, but computers lack this ability.

3. Addressing Ambiguity in NLP:

- **Word Sense Disambiguation (WSD):**

Techniques used to determine the correct meaning of a word in a given context.

- **Named Entity Recognition (NER):**

Identifying and classifying named entities (people, organizations, locations, etc.) to help resolve ambiguity related to those entities.

- **Coreference Resolution:**

Identifying which phrases in a text refer to the same entities, helping to resolve anaphoric ambiguity.

- **Semantic Parsing:**

Converting natural language into a formal, unambiguous representation that can be easily processed by computers.

- **Contextual Analysis:**

Analyzing the surrounding text to determine the intended meaning.

4. Importance of Ambiguity in NLP:

- While ambiguity poses challenges, it is also a crucial aspect of human language and communication.
- Understanding and handling ambiguity is essential for building more robust and intelligent NLP systems.
- Different types of ambiguity require different approaches for resolution, highlighting the complexity of the problem.
-

DYNAMIC PROGRAMMING PARSING IN NLP WITH EXAMPLE

Dynamic programming is a powerful algorithmic paradigm utilized in Natural Language Processing (NLP) for efficient parsing, particularly with context-free grammars (CFGs). It addresses the problem of exponential search spaces by breaking down complex problems into smaller, overlapping subproblems and storing the results of these subproblems to avoid redundant computations. This approach, often implemented as "chart parsing," significantly improves parsing efficiency.

Key Concepts:

- **Overlapping Subproblems:**

The parsing of a sentence often involves analyzing the same substrings multiple times as part of different larger structures. Dynamic programming stores the results of parsing these substrings, preventing recalculation.

- **Optimal Substructure:**

The optimal parse for a sentence can be constructed from the optimal parses of its constituent substrings.

Example: CYK Algorithm for Context-Free Grammar Parsing

The Cocke-Younger-Kasami (CYK) algorithm is a classic example of dynamic programming applied to parsing. It determines if a given sentence can be generated by a CFG in Chomsky Normal Form (CNF) and, if so, constructs all possible parse trees.

How it works:

- **Chomsky Normal Form (CNF):**

The grammar rules must be in CNF, meaning each rule takes one of two forms:

○

$A \rightarrow BC$ or $A \rightarrow a$

$$A \rightarrow BC$$

(where A, B, C are non-terminals)

○

$$A \rightarrow a \text{cap } A \text{ right arrow } a$$

$$A \rightarrow a$$

(where A is a non-terminal and 'a' is a terminal)

- **Chart Initialization:**

A 2D table (chart) is created. Each cell $\text{chart}[i][j]$ will store the non-terminals that can generate the substring from index i to j of the input sentence.

- **Filling the Chart (Bottom-Up):**

- **Length 1 Substrings:** For each word in the sentence, the corresponding $\text{chart}[i][i]$ cell is filled with the non-terminals that can directly generate that word (lexical rules).
- **Longer Substrings:** The algorithm then iteratively fills the chart for increasingly longer substrings. For a substring from i to j, it considers all possible split points k within that substring. If there's a rule

$$A \rightarrow BC \text{cap } A \text{ right arrow cap } B \text{ cap } C$$

$$A \rightarrow BC$$

where

$$B \text{cap } B$$

$$B$$

can generate substring[i...k] and

$C \cap C$

C

can generate substring $[k+1 \dots j]$, then

$A \cap A$

A

is added to $\text{chart}[i][j]$.

- **Result Extraction:**

After filling the entire chart, if the starting symbol of the grammar is present in $\text{chart}[0][n-1]$ (where n is the length of the sentence), then the sentence can be parsed by the grammar. The chart also contains the necessary information to reconstruct the parse trees.

Example:

Consider the sentence "the cat" and a simple CNF grammar:

$S \rightarrow NP VP$
 $S \rightarrow NP VP$

$NP \rightarrow Det N$
 $NP \rightarrow Det N$

$Det \rightarrow the$
 $Det \rightarrow the$

$N \rightarrow cat$
 $N \rightarrow cat$

The CYK algorithm would build a chart, starting with "the" and "cat" mapping to Det and N respectively. Then, it would combine Det and N to form NP, and so on, until it determines if S can generate "the cat".

SHALLOW PARSING

Shallow parsing, also known as chunking or light parsing, is a Natural Language Processing (NLP) technique that identifies and labels segments of a sentence, focusing on extracting meaningful phrases or constituents without performing a full syntactic analysis. Unlike deep parsing, which aims to construct a complete parse tree representing the entire grammatical structure and dependencies within a sentence, shallow parsing focuses on identifying high-level syntactic units.

Key Components of Shallow Parsing:

- **Tokenization:**

The initial step involves breaking down the sentence into individual words or tokens.

- **Part-of-Speech (POS) Tagging:**

Each token is then assigned its corresponding part of speech (e.g., noun, verb, adjective).

- **Chunking:**

This is the core of shallow parsing, where contiguous sequences of tokens are grouped into non-overlapping syntactic chunks, such as:

- **Noun Phrases (NP):** Groups of words functioning as a noun (e.g., "the big dog").
- **Verb Phrases (VP):** Groups of words functioning as a verb (e.g., "is running quickly").
- **Prepositional Phrases (PP):** Phrases starting with a preposition (e.g., "in the garden").

Example:

Consider the sentence: "The quick brown fox jumps over the lazy dog." tokenization.

Code

```
["The", "quick", "brown", "fox", "jumps", "over", "the", "lazy", "dog", "."]
```

POS Tagging.

Code

```
[("The", "DT"), ("quick", "JJ"), ("brown", "JJ"), ("fox", "NN"),  
 ("jumps", "VBZ"), ("over", "IN"), ("the", "DT"), ("lazy", "JJ"),  
 ("dog", "NN"), (".", ".")]
```

Chunking (Shallow Parsing Output).

Code

[NP: The quick brown fox]
[VP: jumps]
[PP: over the lazy dog]

In this example, shallow parsing identifies the noun phrase "The quick brown fox," the verb phrase "jumps," and the prepositional phrase "over the lazy dog." It does not delve into the internal structure of these phrases or the dependencies between words across different phrases, which would be the domain of deep parsing.

Probabilistic CFG in nlp

A Probabilistic Context-Free Grammar (PCFG) is an extension of a Context-Free Grammar (CFG) that assigns probabilities to its production rules. While a CFG defines the set of valid syntactic structures in a language, a PCFG quantifies the likelihood of different structures, allowing for the resolution of ambiguity by favoring more probable interpretations.

Key Components of a PCFG:

- **Non-terminals (N):** Symbols representing grammatical categories (e.g., S for Sentence, NP for Noun Phrase, VP for Verb Phrase).
- **Terminals (T):** The actual words or lexical items of the language.
- **Start Symbol (S):** The non-terminal from which all derivations begin.
- **Production Rules (R):** Rules of the form

$A \rightarrow \beta$ cap A right arrow beta

$A \rightarrow \beta$

, where A is a non-terminal and

β beta

β

is a sequence of non-terminals and/or terminals.

- **Probabilities (P):** Each production rule is associated with a probability,

$P(A \rightarrow \beta) \cap P(\text{open paren} \cap A \text{ right arrow } \beta \text{ close paren})$

$P(A \rightarrow \beta)$

, such that the sum of probabilities for all rules expanding the same non-terminal A equals 1.

Example:

Consider a simplified PCFG for a fragment of English:

Code

```
S -> NP VP (1.0)
NP -> Det Noun (0.6)
NP -> NP PP (0.4)
VP -> Verb NP (1.0)
PP -> Pre NP (1.0)
Det -> 'the' (0.5)
Det -> 'a' (0.5)
Noun -> 'man' (0.5)
Noun -> 'dog' (0.5)
Verb -> 'saw' (1.0)
Pre -> 'with' (1.0)
```

Code copied!

In this example:

- The rule `S -> NP VP` has a probability of 1.0, meaning it's the only way to expand an `S`.
- The `NP` non-terminal can be expanded in two ways: `Det Noun` or `NP PP`. The probabilities (0.6 and 0.4, respectively) indicate that `Det Noun` is more likely.
- The sum of probabilities for rules with `NP` on the left-hand side (0.6 + 0.4) is 1.0.

Resolving Ambiguity with PCFG:

For an ambiguous sentence like "The man saw the dog with the telescope," a CFG would generate two valid parse trees, one where "with the telescope" modifies "dog" and another where it modifies "saw." A PCFG would assign a probability to each of these parse trees by multiplying the probabilities of the rules used in their derivation. The parse tree with the higher overall probability would be considered the most likely interpretation. This probabilistic approach allows for more robust and accurate parsing in Natural Language Processing.

Probabilistic CYK (Cocke-Kasami-Younger) is an extension of the classic CYK algorithm used in Natural Language Processing for parsing sentences based on a Probabilistic Context-Free Grammar (PCFG). While the original CYK algorithm determines if a sentence can be generated by a given Context-Free Grammar (CFG), Probabilistic CYK additionally calculates the probability of different parse trees for a sentence, allowing for the selection of the most likely parse in cases of ambiguity.

Key Concepts:

- **Probabilistic Context-Free Grammar (PCFG):**

A PCFG is a CFG where each production rule is assigned a probability. The sum of probabilities for all rules expanding from a given non-terminal must equal 1. For example:

- $S \rightarrow NP VP$ (prob=0.8)
- $S \rightarrow Aux NP VP$ (prob=0.2)
- $NP \rightarrow Det N$ (prob=0.7)
- $NP \rightarrow ProperN$ (prob=0.3)

- **Chomsky Normal Form (CNF):**

Both standard and probabilistic CYK require the grammar to be in Chomsky Normal Form, where all production rules are of the form $A \rightarrow BC$ (two non-terminals) or $A \rightarrow a$ (a single terminal).

How Probabilistic CYK Works:

The algorithm constructs a table (similar to the standard CYK) where each cell $chart[i][j]$ represents a substring from word i to word j . Instead of just storing the non-terminals that can derive that substring, each cell $chart[i][j]$ stores a list of (Non-terminal, Probability) pairs.

Steps:

- **Initialization:**

For each word w_i in the sentence, fill the diagonal cells $chart[i][i]$ with $(X, prob)$ for all rules $X \rightarrow w_i$.

- **Filling the Table:**

For increasing lengths of substrings (from 2 up to the sentence length):

- For each substring $w_i \dots w_j$:
- Consider all possible binary splits $w_i \dots w_k \mid w_{k+1} \dots w_j$.
- For each split, if there's a rule $A \rightarrow BC$ in the PCFG, and $(B, prob_B)$ is in $chart[i][k]$ and $(C, prob_C)$ is in $chart[k+1][j]$, then add $(A, prob_A * prob_B * prob_C)$ to $chart[i][j]$. If multiple rules can derive A or multiple paths lead to the same A , store the maximum probability or all probabilities for later analysis.

- **Result:**

The cell $chart[0][n-1]$ (for a sentence of length n) will contain the root non-terminal (e.g., s) along with the probabilities of different possible parses for the entire sentence. The parse with the highest probability is considered the most likely.

Example:

Consider the sentence "the man" and a simplified PCFG: $S \rightarrow NP VP$ (prob=0.8), $NP \rightarrow Det N$ (prob=0.7), $Det \rightarrow the$ (prob=1.0), and $N \rightarrow man$ (prob=1.0).

The Probabilistic CYK algorithm would build a table. For "the man":

- $\text{chart}[0][0]$ (for "the"): (Det, 1.0)
- $\text{chart}[1][1]$ (for "man"): (N, 1.0)
- $\text{chart}[0][1]$ (for "the man"):
 - Consider split the | man.
 - Rule $\text{NP} \rightarrow \text{Det N}$.
 - $\text{Probability} = \text{prob}(\text{NP} \rightarrow \text{Det N}) * \text{prob}(\text{Det} \rightarrow \text{the}) * \text{prob}(\text{N} \rightarrow \text{man})$
 - $\text{Probability} = 0.7 * 1.0 * 1.0 = 0.7$
- So, $\text{chart}[0][1]$ would contain (NP, 0.7).

This process allows for ranking parse trees by their probability, which is crucial for disambiguation in real-world NLP applications.

Probabilistic Lexicalized CFGs

Probabilistic Lexicalized Context-Free Grammars (LCFGs) are an extension of Probabilistic Context-Free Grammars (PCFGs) used in Natural Language Processing (NLP) for parsing. While PCFGs assign probabilities to context-free grammar rules, LCFGs incorporate lexical information, specifically the "head" word of a phrase, into the grammar rules and their associated probabilities. This allows LCFGs to capture more fine-grained linguistic information, particularly regarding dependencies and ambiguities that PCFGs might miss.

Key Features:

- **Lexicalization:**

Each non-terminal symbol in an LCFG rule is associated with a head word, which is typically the most "important" or central word within the phrase represented by that non-terminal. For example, in a Noun Phrase (NP), the head is usually the noun.

- **Conditioned Probabilities:**

The probabilities of LCFG rules are conditioned not only on the left-hand side non-terminal but also on its head word and potentially the head words of its children. This allows the model to learn that certain head words are more likely to appear with specific grammatical constructions.

Example:

Consider the sentence "The child ate the cake with a fork."

In a standard PCFG, a rule like $\text{VP} \rightarrow \text{V NP PP}$ would simply have a probability $P(\text{V NP PP} \mid \text{VP})$.

In a Lexicalized PCFG, this rule might become:

VP(ate) -> V(ate) NP(cake) PP(fork)

Here, *ate* is the head of the VP and *V*, *cake* is the head of the NP, and *fork* is the head of the PP. The probability would be conditioned on these head words, for instance:

$P(V(\text{ate}) NP(\text{cake}) PP(\text{fork}) \mid VP(\text{ate}))$

This more specific rule allows the model to differentiate between various verb phrases based on their head words, potentially resolving ambiguities like prepositional phrase attachment (e.g., "ate the cake with a fork" vs. "ate the cake with a friend"). The LCFG can learn that "fork" is a more likely instrument for "eating" than "friend," leading to a higher probability for the correct parse.

Feature structures

In Natural Language Processing (NLP), feature structures, also known as attribute-value matrices (AVMs), are a way to represent linguistic information as a set of attribute-value pairs. They are used in various grammar formalisms (e.g., Head-Driven Phrase Structure Grammar, Lexical Functional Grammar) to capture the properties of words, phrases, and sentences.

Structure:

A feature structure is essentially a collection of attributes, each associated with a specific value. Values can be:

- **Atomic:** Simple, indivisible values (e.g., singular, noun).
- **Complex:** Other feature structures, allowing for nested information (e.g., agreement might be a feature structure containing number and person features).

Example:

Consider the noun phrase "the cats." A feature structure for this phrase might represent its grammatical properties as follows:

Code

```
[  
  CATEGORY: NP,  
  AGREEMENT: [  
    NUMBER: PLURAL,  
    PERSON: 3RD  
  ]  
]
```

In this example:

- CATEGORY and AGREEMENT are attributes.
- NP is an atomic value for CATEGORY.

- The value of AGREEMENT is a complex feature structure containing the attributes NUMBER and PERSON with their respective atomic values PLURAL and 3RD.

Purpose:

Feature structures enable:

- **Representing rich linguistic information:** They can capture morphological, syntactic, and semantic details.
- **Handling agreement and dependencies:** By linking features across different parts of a sentence, they facilitate the modeling of grammatical agreement (e.g., subject-verb agreement).
- **Supporting grammatical rules:** Formalisms use operations like unification on feature structures to combine and check the compatibility of linguistic units during parsing.

Unification of feature structures

In natural language processing (NLP), feature structures are used to represent linguistic information, and unification is a core operation for combining these structures. Unification merges compatible feature structures, resulting in a new structure that contains all the information from the originals, or it fails if the structures contain conflicting information. This process is crucial for parsing, grammar checking, and other NLP tasks where combining linguistic data is necessary.

Feature Structures:

- Feature structures represent linguistic information using feature-value pairs.
- Features can represent categories like "part-of-speech" (POS), "number", "gender", etc., and their values can be atomic symbols or other feature structures.
- For example, a noun phrase might have features for its part-of-speech ("noun"), number ("singular"), and case ("nominative").

Unification:

- Unification combines two feature structures into a single, more specific structure.
- It's a "partial" operation, meaning it might fail if the structures contain contradictory information.
- **Example:** If we have two feature structures:
 - FS1 = [cat: noun, number: singular]
 - FS2 = [number: singular, case: nominative]
 - Unifying them results in FS_unified = [cat: noun, number: singular, case: nominative].

- **Conflict Example:** If FS1 is [number: singular] and FS2 is [number: plural], unification would fail because "singular" and "plural" are incompatible.
- **Variable Example:** If a feature has a variable value (e.g., [number: ?x]), it can be unified with any other value for that feature. If FS1 is [number: ?x] and FS2 is [number: singular], then FS_unified would be [number: singular].

Unification in NLP:

- **Grammar Rules:**

Unification is used to implement grammar rules. For example, a grammar rule like "S -> NP VP" can be represented as a feature structure where the features of the mother node (S) are derived from the features of the daughter nodes (NP and VP) through unification, according to the University of Illinois.

- **Parsing:**

During parsing, unification is used to combine the feature structures of words and phrases to build larger syntactic structures.

- **Agreement:**

Unification ensures agreement between different parts of a sentence, like subject-verb agreement (e.g., ensuring that the number and person features of the subject and verb match).

- **Semantic Interpretation:**

Unification can also be used to combine semantic information from different parts of a sentence.

In summary, unification is a fundamental operation in NLP for combining linguistic information represented in feature structures. It allows for the efficient and consistent representation and manipulation of linguistic data, playing a vital role in tasks like parsing, grammar checking, and semantic interpretation.

UNIT IV SEMANTICS AND PRAGMATICS

Requirements for representation, First-Order Logic, Description Logics – Syntax-Driven Semantic analysis, Semantic attachments – Word Senses, Relations between Senses, Thematic Roles, selectional restrictions – Word Sense Disambiguation, WSD using Supervised, Dictionary & Thesaurus, Bootstrapping methods – Word Similarity using Thesaurus and Distributional methods.

In NLP, semantics focuses on the literal meaning of words and sentences, while pragmatics deals with the meaning derived from context, including speaker intent and situation. Semantics analyzes the meaning of individual words and how they combine to form sentence meaning, whereas pragmatics examines how meaning is influenced by the context of the communication.

Example:

Consider the sentence: "The pitcher is on the table."

- **Semantics:**

A semantic analysis would focus on the literal meaning of the words. "Pitcher" could refer to a container for liquids or a baseball player. "On" indicates a spatial relationship. "Table" is a piece of furniture. The sentence implies a relationship between a pitcher and a table.

- **Pragmatics:**

A pragmatic analysis would consider the context. If the sentence is spoken during a baseball game, "pitcher" is likely referring to the baseball player, and the sentence might be interpreted as a request to move the pitcher away from the field. If the sentence is spoken in a kitchen, "pitcher" likely refers to the container, and the sentence might be interpreted as stating the current location of the pitcher.

Key Differences:

- **Focus:**

Semantics focuses on the inherent meaning of words and sentences, while pragmatics focuses on how meaning is derived from the context.

- **Scope:**

Semantic analysis is generally limited to the sentence level, while pragmatic analysis considers the broader context, including the speaker, listener, and situation.

- **Examples:**

- **Semantics:** Identifying synonyms, antonyms, or hyponyms (e.g., "big" and "huge" are synonyms).

- **Pragmatics:** Understanding sarcasm, indirect requests, or the social implications of language.

In NLP, both semantics and pragmatics are crucial for tasks like:

- **Question answering:**

Understanding the intent behind a question and finding the correct answer, even if the question is phrased in a non-literal way.

- **Machine translation:**

Ensuring that the translated text accurately reflects the meaning of the original text in its context.

- **Chatbots:**

Enabling chatbots to understand user intent and respond appropriately, even if the user's language is ambiguous or indirect.

Requirements for representation

In NLP, representation refers to how textual data is converted into a numerical format that machine learning models can understand and process. This is necessary because most machine learning models operate on numerical data. Effective text representation captures the semantic meaning of words and phrases, allowing machines to analyze and understand text.

Here's a breakdown of key requirements and examples:

1. Requirement: Numerical Representation:

- NLP models require numerical input. Therefore, text must be converted into numerical vectors.

2. Requirement: Semantic Meaning:

- The representation should capture the meaning of words and phrases. Similar words should have similar vector representations.
- For example, "king" and "queen" should be closer in vector space than "king" and "dog".

3. Requirement: Handling Vocabulary:

- NLP models need to handle large vocabularies. Representations should be efficient and scalable.
- Different techniques exist for handling rare words and new vocabulary.

4. Requirement: Efficiency:

- The representation should be computationally efficient for model training and inference.
- Some techniques like Bag of Words (BoW) are simple and efficient but may lose some semantic information.
- More complex methods like word embeddings (Word2Vec, GloVe) capture more semantic meaning but require more computation.

Examples of Text Representation Techniques:

• One-Hot Encoding:

Each word is represented by a binary vector where only one element is 1 (representing the word) and the rest are 0.

- Example: If the vocabulary is { "the", "cat", "sat", "on", "mat" }, the word "cat" might be represented as.
- This is simple but not very efficient for large vocabularies.

• Bag of Words (BoW):

Counts the occurrences of each word in a document. The order of words is ignored.

- Example: For the sentence "The cat sat on the mat", BoW representation might be { "the": 2, "cat": 1, "sat": 1, "on": 1, "mat": 1 }.
- Simple and widely used, but loses word order and some semantic information.

• Word Embeddings (Word2Vec, GloVe):

These techniques learn vector representations that capture semantic relationships between words.

- Example: Word2Vec might represent "king" as a vector, and "queen" as a similar vector, while "dog" would have a different vector.
- These are more complex but capture semantic information well.
- **TF-IDF:**

Term Frequency-Inverse Document Frequency. Weights words based on their frequency in a document and their rarity across the corpus.

- Useful for identifying important words in a document.

- **FastText:**

Breaks words into n-grams and uses these to represent words. Good for handling rare or out-of-vocabulary words.

In essence, the requirements for text representation in NLP boil down to finding a way to encode textual data into a numerical format that is both semantically meaningful and computationally efficient for machine learning models to process.

FIRST-ORDER LOGIC, DESCRIPTION LOGICS

First-Order Logic (FOL) and Description Logics (DLs) are formal systems used in Artificial Intelligence, particularly in Natural Language Processing (NLP), for knowledge representation and reasoning.

First-Order Logic (FOL) in NLP:

FOL, also known as predicate logic, provides a powerful framework to represent knowledge about objects, their properties, and relationships. It extends propositional logic by introducing:

- **Constants:** Specific objects (e.g., John, London).
- **Variables:** Symbols representing unspecified objects (e.g., x, y).
- **Predicates:** Define properties or relationships (e.g., Person(John), Likes(John, Mary)).
- **Functions:** Map objects to other objects (e.g., FatherOf(John)).
- **Quantifiers:** Universal ($\forall$, "for all") and Existential ($\exists$, "there exists") to express statements about collections of objects.

Example in NLP:

Consider the sentence: "Every student studies a subject."

In FOL, this can be represented as:

Code

$$\forall x (\text{Student}(x) \rightarrow \exists y (\text{Subject}(y) \wedge \text{Studies}(x, y)))$$

This formula states that for every x such that x is a Student, there exists a y such that y is a Subject and x Studies y.

Description Logics (DLs) in NLP:

DLs are a family of decidable fragments of FOL, designed specifically for representing and reasoning about conceptual knowledge. They focus on defining classes (concepts) and properties (roles) and the relationships between them, often used for building ontologies. DLs offer a balance between expressiveness and computational tractability.

Example in NLP:

Consider defining "Italian Restaurant" in a DL:

Code

$\text{ItalianRestaurant} \equiv \text{Restaurant} \sqcap \exists \text{ serves. ItalianCuisine}$

This DL axiom states that an ItalianRestaurant is equivalent to a Restaurant that serves at least one ItalianCuisine.

DLs are used in NLP for:

- **Ontology construction:**
Building structured knowledge bases that capture the meaning of terms and their relationships.
- **Semantic parsing:**
Converting natural language sentences into formal representations that can be reasoned with.
- **Question answering:**
Using structured knowledge to answer complex questions posed in natural language.
- **Information extraction:**
Identifying and extracting specific entities and relationships from text based on predefined concepts.

Syntax-driven semantic analysis

Syntax-driven semantic analysis in Natural Language Processing (NLP) refers to the process of deriving the meaning of a sentence or text by leveraging its grammatical structure, or syntax. It involves analyzing how words are arranged and related to each other in a sentence to understand the roles and relationships of entities and actions within that sentence.

Process:

- **Syntactic Parsing:**
The first step involves parsing the sentence to create a syntactic representation, typically a parse tree or a dependency tree. This representation explicitly shows the grammatical relationships between words, such as subject-verb, verb-object, noun-modifier, etc.
- **Semantic Role Assignment:**
Once the syntactic structure is established, semantic roles (e.g., Agent, Patient, Instrument, Location) are assigned to the constituents of the sentence based on their syntactic positions and

the lexical properties of the words (especially verbs). For instance, the subject of an active verb often corresponds to the Agent, while the direct object corresponds to the Patient.

- **Meaning Representation:**

The assigned semantic roles and relationships are then used to construct a formal meaning representation, such as a logical form (e.g., First-Order Predicate Calculus) or a frame-based representation, which captures the core meaning of the sentence in a structured and unambiguous way.

Example:

Consider the sentence: "The cat chased the mouse."

- **Syntactic Parsing:** A parser would identify "cat" as the subject, "chased" as the verb, and "mouse" as the direct object.

Code

```
(S
  (NP (DT The) (NN cat))
  (VP (VBD chased)
    (NP (DT the) (NN mouse)))
  (. .))
```

- **Semantic Role Assignment:** Based on the syntactic structure and the verb "chased," the system would assign semantic roles:

- "cat" is identified as the Agent (the one performing the action).
- "mouse" is identified as the Patient (the one undergoing the action).
- "chased" represents the Action.

- **Meaning Representation:** This can be represented in a logical form:

Code

```
chase(Agent: cat, Patient: mouse)
```

This example demonstrates how the syntactic structure provides the necessary scaffolding to correctly interpret the semantic roles and understand the meaning of the sentence, distinguishing who performed the action and who was affected by it.

.

.

In natural language processing (NLP), semantic attachments refer to the process of connecting words and phrases in a sentence to their corresponding meanings and relationships within a specific context. This involves representing the meaning of words by linking them to concepts or entities they refer to, and understanding how these words relate to each other semantically.

Example:

Consider the sentence "I saw a bank." The word "bank" has multiple meanings: a financial institution or the side of a river.

Semantic attachments would represent this as follows:

- **"Bank" (financial institution):**
 - Word: "bank"
 - Sense: "financial institution"
 - Attachment: Links to concepts like "account," "loans," or "mortgages".
 - **"Bank" (river bank):**
 - Word: "bank"
 - Sense: "river bank"
 - Attachment: Links to concepts like "water," "shore," or "sediment".
- By using semantic attachments, NLP systems can better understand the nuances of language and resolve ambiguities, which is crucial for various tasks like information retrieval, machine translation, and sentiment analysis.

Key aspects of semantic attachments:

- **Lexical semantics:**
Analyzing the meanings of individual words.
- **Semantic relations:**
Identifying relationships between words (e.g., hyponymy, homonymy, polysemy).
- **Meaning representation:**
Creating a representation of the sentence's meaning, often using first-order predicate logic (FOPC).
- **Contextual understanding:**
Resolving ambiguities and determining the correct meaning based on context.

In natural language processing (NLP), word sense refers to the different meanings a word can have, and word sense disambiguation (WSD) is the task of identifying which meaning is intended in a specific context. This is crucial because many words have multiple senses, and understanding the correct sense is vital for accurate NLP tasks like machine translation, question answering, and information retrieval.

Word Sense

A word sense is a distinct representation of one aspect of a word's meaning. For example, the word "bank" can refer to a financial institution or the land alongside a river. These different meanings are called word senses.

Word Sense Disambiguation (WSD)

WSD is the process of determining the correct sense of a word in a given context. It's a core NLP problem because computers struggle to understand the nuances of human language and may misinterpret words with multiple meanings.

Examples

- **Polysemy:**

The word "bass" can refer to a type of fish or a musical instrument. WSD would aim to identify which sense is intended in a sentence like "He caught a bass" (fish) or "She plays the bass" (instrument).

- **Homonymy:**

Words like "may" can be a verb (e.g., "I may go") or a month (e.g., "May is a beautiful month"). WSD helps distinguish these senses.

- **Contextual clues:**

In the sentence, "The company is looking for a new CEO," the word "company" likely refers to a business entity, while in "The friends enjoyed the company of each other," it refers to companionship.

Importance of WSD

WSD is essential for various NLP applications:

- **Machine Translation:** Accurately translating a word requires understanding its intended meaning in the source language.
- **Question Answering:** WSD helps systems understand the user's query and provide relevant answers.
- **Information Retrieval:** WSD improves search results by understanding the user's intent.

Methods for WSD

Various techniques are used for WSD, including:

- **Knowledge-based methods:** Rely on dictionaries and lexical resources like WordNet.
- **Supervised methods:** Train on manually sense-tagged data.

- **Unsupervised methods:** Discover sense distinctions from unlabelled data.
- **Transformer-based models:** Modern approaches like BERT contextualize word meanings dynamically.

In essence, WSD is a challenging but vital NLP task that bridges the gap between the surface form of words and their underlying meanings, enabling computers to better understand and process human language

Relations between Senses

In Natural Language Processing (NLP), "senses" refer to the distinct meanings or interpretations a word can have, and "relations between senses" describe the systematic connections that exist between these different meanings. Understanding these relations is crucial for tasks like Word Sense Disambiguation (WSD), where the goal is to identify the correct meaning of a word in a given context.

Types of Relations between Senses:

- **Polysemy:**
This occurs when a single word has multiple related senses.
 - **Example:** The word "bank" can refer to a financial institution ("I deposited money at the bank") or the land alongside a river ("We sat on the river bank"). These senses are distinct but related through the concept of a "side" or "edge."
 - **Homonymy:**
This describes words that are spelled and/or pronounced the same but have unrelated meanings.
 - **Example:** "Bat" can refer to a flying mammal or a piece of sports equipment. The senses are not inherently connected.
 - **Metonymy:**
This is a specific type of polysemy where one aspect of a concept or entity is used to refer to another aspect or the entity itself.
 - **Example:** "The White House" often refers to the U.S. administration, not just the building itself. Here, the building (a place) stands for the organization/people associated with it.
 - **Synonymy:**
This describes words or senses that have the same or very similar meanings.
 - **Example:** The sense of "home" as a "structure/building" is synonymous with a sense of "house."
 - **Antonymy:**
This describes senses that have opposite meanings.
 - **Example:** "Hot" and "cold" are antonyms.
- Example in WSD:

Consider the sentence: "He went to the bank to fish."

To disambiguate the sense of "bank," an NLP system would analyze the surrounding words (context). The presence of "fish" strongly suggests the "river bank" sense, as fishing is typically associated with bodies of water, not financial institutions. This demonstrates how the relationship between the sense of "bank" and the activity of "fishing" helps in selecting the correct meaning. Modern WSD algorithms often leverage these semantic relationships, explicitly or implicitly, through techniques like knowledge-based methods (e.g., Lesk algorithm) or supervised learning using contextual features.

Thematic Roles

In natural language processing (NLP), thematic roles, also known as semantic roles or theta roles, define the relationship between a noun phrase and the verb in a sentence. They describe how a noun phrase participates in the event or state expressed by the verb. For example, in the sentence "The dog chased the cat," "the dog" is the Agent (the one performing the action), and "the cat" is the Patient (the one undergoing the action).

Here's a breakdown of common thematic roles with examples:

1. Agent: The entity that performs the action. Example: "The chef cooked the meal." (The chef is the agent).
2. Patient: The entity that is affected by the action. Example: "The chef cooked the meal." (The meal is the patient).
3. Instrument: The entity used to perform the action. Example: "The chef cooked the meal with a spoon." (The spoon is the instrument).
4. Experiencer: The entity that experiences a feeling, perception, or state. Example: "The child saw the movie." (The child is the experiencer).
5. Source: The starting point of a movement or action. Example: "The bird flew from the nest." (The nest is the source).
6. Goal: The ending point of a movement or action. Example: "The bird flew to the tree." (The tree is the goal).
7. Location: The place where the action occurs. Example: "The students are studying in the library." (The library is the location).
8. Beneficiary: The entity that benefits from the action. Example: "He baked a cake for his friend." (His friend is the beneficiary).

Thematic roles are crucial for understanding the meaning of sentences beyond surface-level syntax. They help in tasks like:

- **Semantic Role Labeling (SRL):** Identifying the thematic roles of noun phrases in a sentence.
- **Machine Translation:** Ensuring accurate translation by preserving the underlying semantic relationships.
- **Question Answering:** Understanding the context of questions to provide relevant answers.

- **Information Extraction:** Identifying key entities and their roles in a text.
By analyzing the thematic roles, NLP systems can better understand the relationships between words and phrases, leading to a deeper comprehension of the text.

selectional restrictions

In Natural Language Processing (NLP), selectional restrictions are semantic rules that govern which words can appropriately combine with others based on their meanings. These restrictions help determine if a sentence is semantically plausible by ensuring that the relationships between words are logical and consistent with our understanding of the world.

Here's a breakdown:

- **What they are:**
Selectional restrictions are like filters that check if the words in a sentence fit together based on their inherent meanings and the roles they play (subject, object, etc.).
- **Why they matter:**
They prevent nonsensical or grammatically incorrect sentences. For example, "The building drank coffee" violates the selectional restriction of "drink," which typically requires an animate subject.
- **How they work:**
Verbs, for instance, often have specific requirements for their subjects and objects. The verb "eat" usually needs an animate subject and an edible object.
- **Example:**
"The dog ate the bone" is a valid sentence because it adheres to the selectional restrictions of "eat". However, "The dog ate the building" is a selectional restriction violation because buildings are not edible.
- **Beyond verbs:**
These restrictions apply to other word types as well, like adjectives and nouns, ensuring that they make sense in context.
In essence, selectional restrictions help NLP systems understand and interpret language by ensuring that the relationships between words are semantically coherent.

Word Sense Disambiguation (WSD) in Natural Language Processing (NLP) is the computational process of identifying the correct meaning or "sense" of a word when it appears in a particular context, especially when that word has multiple possible meanings (polysemy).

Example:

Consider the word "bank" in the following sentences:

- "I went to the bank to deposit money."
- "The river overflowed its banks."

In the first sentence, "bank" refers to a financial institution. In the second sentence, "bank" refers to the land alongside a river. A human readily understands the distinction based on the surrounding words and context. However, for a computer, these are simply instances of the word "bank." WSD aims to equip NLP systems with the ability to make this distinction.

How it works (simplified):

WSD algorithms typically analyze the context of the ambiguous word, often by examining the words immediately surrounding it, the grammatical relationships, or by comparing the context with pre-defined sense definitions from resources like WordNet. For instance, in the "bank" example, the presence of words like "deposit" and "money" in the first sentence strongly suggests the financial institution sense, while "river" and "overflowed" in the second sentence point to the riverbank sense.

Supervised Word Sense Disambiguation (WSD)

Supervised Word Sense Disambiguation (WSD) in Natural Language Processing (NLP) involves training a machine learning model on a labeled dataset to determine the correct meaning of a word in a given context. The labeled dataset contains instances of polysemous words (words with multiple meanings) along with their assigned sense based on the surrounding text. The model learns to associate specific contextual features with particular senses, enabling it to disambiguate the meaning of the same word in new, unseen contexts.

Example:

Consider the word "bank," which can refer to a "financial institution" or the "edge of a river." Training Data Preparation.

A dataset is created where each instance of "bank" is manually labeled with its correct sense.

- "I deposited money at the bank." (Sense: financial institution)
- "The boat was tied to the bank of the river." (Sense: edge of a river)
- "She works at a large investment bank." (Sense: financial institution)

Feature Extraction.

From this labeled data, features are extracted to represent the context of the target word. Common features include:

- **Collocational features:** Words immediately surrounding the target word (e.g., "river," "money," "deposited").
 - **Part-of-speech tags:** The grammatical category of surrounding words.
 - **Bag-of-words features:** All words in a defined window around the target word, without considering their order.
- Model Training.

A supervised machine learning algorithm, such as Naive Bayes, Support Vector Machines (SVMs), or more recently, deep learning models like BERT, is trained on these features and their corresponding sense labels. The model learns the patterns and relationships between the contextual features and the intended sense of "bank." Disambiguation of New Text.

When a new sentence containing "bank" is encountered, the trained model extracts the same types of features from its context. Based on the learned patterns, the model predicts the most probable sense of "bank" in that specific instance.

- **New Sentence:** "We picnicked by the bank."
- **Model's Prediction:** Based on features like "picnicked by," the model would likely predict the "edge of a river" sense.

Dictionary & Thesaurus,

In NLP, dictionaries and thesauri serve distinct yet complementary roles. Dictionaries provide definitions and grammatical information about words, while thesauri focus on relationships between words, particularly synonyms and related terms. They are both valuable resources for tasks like text analysis, information retrieval, and language generation.

Dictionaries in NLP:

- **Definition and Usage:**
Dictionaries are used to determine the meaning of words in context, often referred to as Word Sense Disambiguation (WSD).
 - **Tokenization:**
They are crucial for breaking down text into individual words (tokenization), especially when dealing with multi-word expressions or named entities.
 - **Example:**
In Named Entity Recognition (NER), a dictionary can identify "New York" as a single entity, even though it consists of two words.
 - **Sentiment Analysis:**
Dictionaries of sentiment-related words help analyze the emotional tone of text.
- Thesauri in NLP:
- **Synonym and Related Word Retrieval:**

Thesauri help find synonyms, related words, and even broader or narrower terms (hypernyms and hyponyms).

- **Contextual Similarity:**

They can help identify words that are contextually similar, which is useful for tasks like information retrieval and text summarization.

- **Example:**

Finding a synonym for "happy" might involve retrieving words like "joyful," "content," or "glad".

- **Improving Text Generation:**

Thesauri can be used to enhance the variety and richness of text by suggesting alternative phrasing.

- **Example:**

In a sentence like "The cat sat on the mat," a thesaurus could suggest "The feline perched on the rug".

Key Differences:

- **Word Senses:**

Dictionaries provide well-defined word senses, while thesauri may focus on broader meaning clusters.

- **Contextual Analysis:**

Dictionaries are essential for disambiguating word meanings based on context, while thesauri are useful for finding related terms within a specific context.

- **NLP Tasks:**

Dictionaries are used for tasks like tokenization, NER, and sentiment analysis, while thesauri are valuable for tasks like text summarization, information retrieval, and text generation.

In essence: Dictionaries offer precise definitions and grammatical information, while thesauri provide broader semantic relationships between words, both contributing to a deeper understanding and manipulation of language in NLP.

Bootstrapping in Natural Language Processing (NLP) refers to a class of methods that leverage a small amount of initial labeled data to iteratively expand a dataset or improve a model, often by using the model's own predictions to generate new training examples. This is particularly useful in NLP where obtaining large amounts of labeled data can be expensive and time-consuming.

How it works:

- **Seed Data:** Start with a small set of manually labeled examples (the "seed" data).
- **Initial Model Training:** Train an initial NLP model (e.g., a classifier) on this seed data.
- **Prediction on Unlabeled Data:** Use the trained model to make predictions on a larger pool of unlabeled data.
- **Selection of High-Confidence Predictions:** Identify predictions where the model is highly confident (e.g., above a certain probability threshold).
- **Adding to Training Data:** Add these high-confidence predictions (along with their predicted labels) to the training set.
- **Retraining:** Retrain the model on the expanded training set.
- **Iteration:** Repeat steps 3-6 until a desired level of performance or data size is reached, or until no significant improvements are observed.

Example in NLP: Named Entity Recognition (NER)

Imagine building an NER system to identify "Person" and "Organization" entities, but you only have a few initial examples.

- **Seed Data:**

You start with sentences like:

- "Barack Obama visited Google." (Barack Obama: Person, Google: Organization)
- "Elon Musk founded Tesla." (Elon Musk: Person, Tesla: Organization)
- **Initial Model:**

Train a simple NER model (e.g., a rule-based system or a basic sequence tagger) on this limited data.

- **Prediction:**

The model then processes unlabeled text like:

- "The CEO of Microsoft, Satya Nadella, announced a new product."
- **High-Confidence Selection:**

The model might confidently predict "Microsoft" as "Organization" and "Satya Nadella" as "Person" based on the patterns learned from the seed data.

- **Adding to Training Data:**

These new examples are added to the training set.

- **Retraining:**

The model is retrained with the expanded data, improving its ability to recognize similar entities in new contexts.

- **Iteration:**

This process continues, potentially identifying more entities and improving the model's robustness over time.

This iterative expansion allows the model to learn from a progressively larger and more diverse dataset without requiring extensive manual annotation for every single example.

In Natural Language Processing (NLP), determining word similarity is crucial for tasks like information retrieval, question answering, and text summarization. Two primary approaches to calculating word similarity are thesaurus-based methods and distributional methods.

1. Thesaurus-Based Methods:

These methods leverage pre-existing lexical resources like thesauruses (e.g., WordNet) that encode semantic relationships between words. Word similarity is determined by analyzing the structure of these resources and the relationships between words within them.

- **Principle:**

Words are considered similar if they are close in the hierarchical or network structure of the thesaurus, or if they share common semantic properties (e.g., being synonyms, hyponyms, or having a shared hypernym).

- **Example (using WordNet):**

To find the similarity between "car" and "automobile," a thesaurus-based method would identify that both are synonyms and belong to the same synset (a set of words with the same meaning). The similarity could also be calculated by finding the shortest path distance between their respective synsets in the WordNet hierarchy, or by considering their shared hypernyms (e.g., "vehicle").

2. Distributional Methods:

These methods operate on the "distributional hypothesis," which states that words appearing in similar contexts tend to have similar meanings. They represent words as vectors in a high-dimensional space, and similarity is calculated based on the proximity of these vectors.

- **Principle:**

Words are considered similar if they frequently appear in similar linguistic contexts within a large corpus of text. This context can be defined by surrounding words, grammatical dependencies, or other features.

- **Example (using Word2Vec):**

To find the similarity between "king" and "queen," a distributional method like Word2Vec would analyze a large text corpus. It would learn that "king" often appears in contexts with words like "throne," "ruler," and "kingdom," and "queen" appears in similar contexts. The resulting word vectors for "king" and "queen" would be close in the vector space, indicating their semantic similarity. The similarity can then be quantified using metrics like cosine similarity between their vectors.

Comparison:

- **Thesaurus-based methods**

rely on expert-curated knowledge and are strong for capturing explicit semantic relations. However, they can be limited by the coverage and granularity of the thesaurus.

- **Distributional methods**

learn relationships directly from data, offering broader coverage and the ability to capture nuanced semantic similarities. Their effectiveness depends heavily on the size and quality of the training corpus.

In practice, a combination of both approaches can yield more robust and comprehensive word similarity measures.

Bootstrapping methods

Bootstrapping in Natural Language Processing (NLP) refers to a set of methods used to expand a small set of labeled data, typically through iterative self-training or semi-supervised learning. The core idea is to leverage a small amount of initial knowledge (e.g., seed words, patterns) to automatically identify more instances of a specific category or relation in a larger unlabeled corpus.

How it works:

-

- **Initialization:**

-

Start with a small set of "seed" examples or patterns. These are manually curated and represent the target concept.

-

-

- **Pattern/Instance Extraction:**

-

Use the current set of seeds to find new patterns or instances in the unlabeled data.

-
- **Pattern-based bootstrapping:** If you have seed instances (e.g., names of companies), you might look for common linguistic patterns surrounding them (e.g., "CEO of [Company Name]").
- **Instance-based bootstrapping:** If you have seed patterns (e.g., "[Person] works for [Organization]"), you might use them to extract new instances of entities (e.g., "John works for Google").
-

Expansion and Refinement:

-
- Add the newly discovered patterns or instances to the seed set. This expanded set is then used in the next iteration to find even more patterns or instances.
-
-

Iteration and Termination:

-
- Repeat steps 2 and 3 until a stopping criterion is met (e.g., no new patterns/instances found, a certain number of iterations completed, or a performance threshold reached).
-

Example in NLP: Named Entity Recognition (NER)

Imagine you want to build a system to identify company names, but you only have a few examples: "Google," "Microsoft," "Apple."

- **Initialization:**
 - **Seed Instances:** {"Google", "Microsoft", "Apple"}
 - **Pattern Extraction (Iteration 1):** Scan a large unlabeled text corpus. You might find patterns like:
 - "[Company Name] announced..."
 - "the CEO of [Company Name]"
 - "...acquired by [Company Name]"

Using your seed instances, you find these patterns surrounding them.

-
- **Instance Extraction (Iteration 1):**

- Now, use these extracted patterns to find new potential company names in the corpus. For example, if you find "the CEO of Tesla," you can infer "Tesla" is a company name.

-
-

Expansion and Refinement:

- Add "Tesla" to your seed instances. Add the new patterns found in step 2 to your set of patterns.

Iteration 2 and beyond:

- Repeat the process. With "Tesla" now in your seed instances, you might find new patterns associated with it, leading to the discovery of more company names, and so on.
- This iterative process helps to automatically grow the dataset and build a more comprehensive system without extensive manual annotation.

Word Similarity using Thesaurus and Distributional methods

Word similarity in Natural Language Processing (NLP) can be determined using two primary approaches: Thesaurus-based methods and Distributional methods.

1. Thesaurus-based Similarity:

This approach leverages pre-existing lexical resources like thesauri or semantic networks (e.g., WordNet) to measure similarity based on explicit relationships between words. These relationships include:

- **Synonymy:** Words with similar meanings (e.g., "happy" and "joyful").
- **Antonymy:** Words with opposite meanings (e.g., "hot" and "cold").

- **Hyponymy/Hypernymy:** Hierarchical relationships where one word is a specific instance of another (e.g., "car" is a hyponym of "vehicle," and "vehicle" is a hypernym of "car").

Example: To find the similarity between "car" and "automobile" using a thesaurus, one would look for direct synonymy links or a shared hypernym that indicates a close semantic relationship.

2. Distributional Similarity (Distributional Semantics):

This approach operates on the principle that "words that appear in similar contexts have similar meanings." It involves representing words as vectors in a high-dimensional space, where the proximity of these vectors indicates semantic similarity. Common techniques include:

-

Count-based methods:

-

These methods count co-occurrences of words within a defined context window (e.g., Pointwise Mutual Information (PMI)).

-
-

Predictive methods (Word Embeddings):

-

These methods learn dense vector representations (embeddings) of words by predicting surrounding words in a given context (e.g., Word2Vec, GloVe).

-

Example: To find the similarity between "cat" and "dog" using distributional methods, a model would analyze a large corpus of text. If "cat" frequently appears in contexts like "a fluffy ____," "chasing a mouse," or "loves to nap," and "dog" appears in similar contexts like "a playful ____," "chasing a ball," or "loves to run," their respective word vectors would be close in the embedding space, indicating high similarity. This similarity is often quantified using metrics like cosine similarity between the word vectors.

